CS161 Homework 6 **Due:** 27 May 2016
Submit on Gradescope **Handed out:** 20 May 2016

Instructions: Please answer the following questions to the best of your ability. If you are asked to design an algorithm, please describe it clearly and concisely, prove its correctness, and analyze its running time. If you are asked to show your work, please include relevant calculations for deriving your answer. If you are asked to explain your answer, give a short ($\sim$ 1 sentence) intuitive description of your answer. If you are asked to prove a result, please write a complete proof at the level of detail and rigor expected in prior CS Theory classes (i.e. 103). When writing proofs, please strive for clarity and brevity (in that order). Cite any sources you reference.

1 (13 points) Poe learns of Karger's Algorithm

(a) (3 points) Poe hears about Karger's algorithm to solve the minimum cut problem for unweighted graphs. Poe now wants to figure out how to solve the minimum cut problem for *weighted* graphs. Now, the size of a cut is measured as the sum of the weights of the edges going across the cut, and Poe wants to minimize this weight sum over all cuts. Barr has devised a secret algorithm that he refuses to share with Poe. Poe goes to Barr's evil twin brother Burr the Bear. Burr tells Poe that he can solve the problem for weighted graphs by replacing each edge of weight k by a path of k edges of unit weight. That is, if there is an edge (u, v) of weight k, then Poe should delete the edge, rename u by $x_{u,v,0}$ and v by $x_{u,v,k}$, add new vertices $x_{u,v,1}, x_{u,v,2}, \ldots, x_{u,v,k-1}$ and add new edges of unit weight between $x_{u,v,i}$ and $x_{u,v,i+1}$ where $i \in [0, k-1]$. After all edges are replaced by paths of unit-weight edges, the resulting graph is an unweighted graph.

Poe can then use Karger's algorithm to solve his problem (assume that Poe runs Karger's algorithm so many times that with probability 1 Poe will find a min-cut in the unweighted graph). The solution will be a minimum cut C (which is a partition of vertices into two non-empty subsets) for the unweighted graph. Poe then remove all vertices $x_{u,v,i}$ for all u, v and $i \in [1, w(u, v) - 1]$ from C to obtain a partition of vertices in the original graph (let us call it C'). Poe hopes that C' is a min-cut in the original, weighted graph, but sadly this scheme fails in many cases.

Find a small counterexample (with at most 5 vertices). (In your example all edge weights should be positive integers.) Your example should show that C' is not a valid cut or is not a min-cut.

Solution:

Let G be a path a-b-c with edge weights $w(a, b) = 1$ and $w(b, c) = 2$. The new graph G' will be a path a-b-x-c with unit weights. One min-cut C in G' is $(\{a, b, x\}, \{c\})$, but the corresponding cut C' in G is $(\{a, b\}, \{c\})$ is not a min-cut in the original graph.

(b) (3 points) The max-cut of a graph is a cut with maximum number of edges crossing the cut. The complement G' of a graph G is defined as a graph on the same set of vertices as G but so that there is an edge (u, v) in G' if and only if there is no edge (u, v) in G (hence complementing the edges). Consider any cut S of G and the same cut S in G'. Poe uses the following logic: the fewer edges there are across S in G, the more non-edges there across S in G and hence the more edges there are across S in G'. Because of this, Poe thinks that given a max-cut of some graph G, it should correspond to a min-cut of the complement of G.

Formally, let $c_{\min}(G)$ be the size of a min-cut in graph G and let $c_{\max}(G)$ be the size of a max-cut in G. If C is a min-cut in graph G (that is, C has $c_{\min(G)}$ edges crossing it in G), then Poe thinks that C is a max-cut in graph G' (that is, C has $c_{\max(G')}$ edges crossing it in G'). Sadly this is not true; that is, C may not be a max-cut in G'. Find a small counterexample (G, which should be a simple graph) with at most 5 vertices. Where does Poe's logic go wrong? (It's sufficient to explain why this logic fails in your example.)

Solution:

Let G be a path a-b-c-d. A min-cut in G is $(\{a\}, \{b, c, d\})$ (call it C). The complement G' is a path b-d-a-c, but C (whose cost is 2) is not a max-cut in G' because there is a cut of size 3 in G'. Poe's logic fails because a min-cut in G and a max-cut in G' may not necessarily have the same number of vertices in each subset.

(c) (7 points) In class we learned Karger's randomized min-cut algorithm as well as the AmplifiedKarger algorithm, which makes independent runs of Karger's algorithm on n vertices. Here we consider another variation of Karger's algorithm, which we call (a, b)-Karger and we will compare it to Karger's algorithm.

(a, b)-Karger first contracts random edges until the multigraph contains a supernodes (the initial graph has n vertices), as Karger's algorithm normally does (except Karger's continues until 2 supernodes remain and here we stop earlier). Then the algorithm makes b independent runs of Karger's algorithm on this multigraph on a supernodes. The algorithm outputs the smallest cut found in any of the b runs.

(i) (1 point) Suppose we run Karger's algorithm twice independently and output the minimum cut found in either iteration. What is the probability of success; that is, with what probability does this algorithm find a min-cut in at least one of those two runs? (As we did in class, provide a lower-bound on the probability of success of this algorithm.)

Solution:

We know each run has probability of success (finding a min-cut) at least $2/n(n-1)$. Finding a min-cut in at least one of the two runs is then at least:

$$1 - \left(1 - \frac{2}{n(n-1)}\right)^2$$

(ii) (3 point) Determine the total number of edge contractions performed by (a, b)-Karger (in terms of n, a, b). Give a lower bound on the probability of finding a minimum cut (in terms of n, a, b). (Assume $a < n$ for simplicity. Also, as we did in analysis of Karger's algorithm, you should lower-bound the probability of finding a specific min-cut for simplicity.)

Solution: The first stage performs $n - a$ contractions. Each run in the second stage performs $a - 2$ contractions. In total we have $n - a + b(a - 2)$ edge contractions.

The probability of success is the probability of a desired min-cut being existent after the first $n-a$ contractions (which is $a(a-1)/n(n-1)$ as we did in lecture) multiplied by the probability of finding a correct min-cut in the smaller graph in b independent rus.

The second probability for each run is $2/a(a-1)$; the probability that all the b independent runs fail is then at most $(1 - 2/a(a-1))^b$; the probability that at least one of b runs is successful (conditioining on the first event mentioned earlier) is at least $1 - (1 - 2/a(a-1))^b$.

This leads to the following answer:

$$\frac{a(a-1)}{n(n-1)}\left(1 - \left(1 - \frac{2}{a(a-1)}\right)^b\right)$$

(iii) (3 points) Let us compare Karger's algorithm and (a, b)-Karger. Running Karger's algorithm twice as in part (i) above, results in $2(n-2)$ edge contractions over both runs. In this problem, we will find values of a, b such that (a, b)-Karger performs $2(n-2)$ edge contractions in total, but its probability of finding a min-cut is better than running Karger's algorithm twice independently.

Fix $a = n^{1/3}$ (it turns out that this setting is optimal). For this setting of a, find the value β of b which leads to $2(n-2)$ edge contractions. Then, using your answer from (ii), give a lower bound on the probability of success of the $(n^{1/3}, \beta)$-Karger algorithm (use Ω to simplify your answer).

(Hint: (a, b)-Karger's probability of success should be better than that of Karger's run twice in Part (i). You may find it useful to simplify your answer to Part (ii) by replacing $(a-1)$ terms by a and $(n-1)$ terms by n, if any.)

Solution: Since $n-a+\beta(a-2) = 2(n-2)$, we have $\beta = (2(n-2)-n+a)/(a-2) = (n+a-4)/(a-2)$.

As hint suggests, we first simplify the answer from above (let p_s denote the probability of success we wish to bound):

$$p_s \geq \frac{a^2}{n^2}\left(1 - \left(1 - \frac{2}{a^2}\right)^b\right)$$

Recall that the probability of success is bounded below by the above quantity. Using $(1-1/x)^x \leq 1/e$ (learned from lecture), we get the following (note that $-(1-1/x)^x \geq -1/e$, so we can apply this to get a lower-bound):

$$p_s \geq \frac{a^2}{n^2}\left(1 - \exp(-2b/a^2)\right)$$

Now we plug in $b = \beta$ and $a = n^{1/3}$:

$$p_s \geq \frac{n^{2/3}}{n^2}\left(1 - \exp\left(\frac{-2(n + n^{1/3} - 4)}{n - 2n^{2/3}}\right)\right)$$

Note that the exp() term converges to $1/e^2$ as $n \to \infty$, so we can state $p_s = \Omega(n^{-4/3})$.
If we run Karger's twice, roughly the probability of success is bounded below by $\Omega(n^{-2})$; $n^{-4/3}$ is much better than n^{-2} asymptotically (as we wish to maximize probability of success).

2 (10 points) Minimum Spanning Trees

In all of the following assume that G is an undirected, connected graph on n nodes with distinct, positive edge weights. Also assume that G is simple (no self-loops and no multi-edges) in all parts.

(a) (2 points) Consider the edges of G sorted in increasing order of weight and place the first $n-1$ edges into a set S. That is, S is the set of $n-1$ edges with the smallest weights.

Prove or disprove: S is a minimum spanning tree in G (assume $n \geq 2$).

(Concisely prove the claim or give a small counterexample.)

Solution:

Consider a graph with four vertices $\{a, b, c, d\}$ and edges $w(a,b) = 1, w(a,c) = 2, w(b,c) = 3, w(a,d) = 4$. S consists of the first three edges which forms a cycle, and thus is not a spanning tree.

(b) (3 points) Prove or disprove: There is a unique minimum spanning tree in G. (Concisely prove the claim or give a small counterexample.)

Solution: There is a unique min-cost spanning tree.

We use the cycle property as lemma: The most expensive edge on any cycle does not belong to MST (when edge weights are distinct).

Let C be any cycle in graph and e be the most expensive edge on G. Suppose T is any spanning tree containing e (we show that T is not an MST then). If we delete e from T', then $T' \setminus \{e\}$ partitions the verices into two connected components (two trees), $(S, V \setminus S)$; if $e = (u, v)$, let S be the set of nodes conneted to u in $T' \setminus \{e\}$ and $V \setminus S$ be the set of nodes connected to v. Since $C \setminus \{e\}$ is a path from u to v (by definition of a cycle), there must be some edge e' crossing the cut; in particular, $w(e) > w(e')$ by assumption, and let $T' = T \setminus \{e\} \cup \{e'\}$ (then T' has smaller cost than T). We claim that T' is a spanning tree. We already know that S and $V \setminus S$ are trees (from T), and for every node $x \in S$ and $y \in V \setminus S$, there is a path from x to u (in S) and y to v (in $V \setminus S$), and we use e' to create a path between x, y. This shows T' is a spanning tree.

To prove the main part:

Let T be any MST in G. We show that T is unique. For any edge $e' \notin T$, $e' \cup T$ contains a cycle C with e' in it; e' must be the most expensive edge on this cycle – otherwise, if there is another edge e'' on C

that is more expensive, then T cannot be MST by the lemama, which is a contradiction. This applies for every edge e' not in T, and we conclude that every edge $e' \notin T$ does not belong to any MST. Thus T is the only MST in G.

(c) (5 points) Let T be a minimum spanning tree of G. Let e be some edge in G (which may or may not belong to T). Let us obtain a new graph G' from G by preserving everything the same except that we decrease the weight of e (but assume that the weights of all edges in G' are still distinct). Design an algorithm that can find a minimum spanning tree T' of G' in time $O(n)$, given G, T, e and its newly decreased weight w. Prove the correctness and analyze the runtime.

Solution:

We use the cycle property as lemma: The most expensive edge on any cycle does not belong to MST.

Let T be an MST of G as stated, and let e be some edge we are reducing the weight of (recall T is unique from previous part). Consider any edge $e' \notin T$; since $T \cup \{e'\}$ induces a cycle containing e', e' must be the most expensive edge on this cycle (or it contradicts the lemma). Note that this holds for all $e' \notin T$.

Now we reduce the weight of e; for every $e' \notin T$ with $e' \neq e$, e' remains to be the most expensive edge on the cycle in $T \cup \{e'\}$ because we are reducing the weight of e which does not affect e' being the most expensive. Therefore, in the new graph, any MST (there is a unique one due to part (a)) cannot contain any $e' \notin T$ with $e' \neq e$. In particular, if $e \in T$, then T remains to be an MST in the new graph, and if $e \notin T$, then an MST must contain edges from $T \cup \{e\}$. In the latter case, $T \cup \{e\}$ creates a cycle containing e; if $e = (u, v)$, the cycle contains e and the u-v path on T. By the cycle property, the most expensive edge in this cycle cannot be in an MST of the new graph.

With this proof, the following algorithm's correctness follows:

Given T, G, and e and its new weight w, check if $e \in T$ or not (we can just scan the edges of T in linear time). If $e \in T$, just output T. If $e \notin T$, let $e = (u, v)$. Using DFS(u) on T, find the u-v path in $O(n)$ time (since T contains $n - 1$ edges, this runs in $O(n)$ time). Scan the u-v path on T to find the most expensive edge e'' on this path in $O(n)$ time; if $w(e'') > w$ then we output $T' = T \cup \{e\} \setminus \{e''\}$ as answer, and if $w(e'') < w$ then we output T.

3 (12 points) Barr Selling Fish

Barr the Bear has started a business to sell fish to Poe and his fellow penguins. The penguin customers submit many fish orders, but Barr can only process one order at a time. Suppose that Barr currently has orders from n penguin customers (label them as $1, 2, \ldots, n$). Customer i's order takes t_i time to complete. Barr is going to process each penguin's order one by one, and the scheduling of orders can be described as a permutation of the customers. Let C_i denote the completion time of order i. For instance, if customer j's order is the first to be completed, then we would have $C_j = t_j$ (assume Barr begins processing orders at time 0); if customer k's order is the second to be completed after that, then $C_k = C_j + t_k = t_j + t_k$, and so on. Each customer is of different importance to Barr's business, and we denote this relative weight by w_i (for customer i). Barr wishes to minimize the weighted sum of the completion times of n orders, $\sum_{i=1}^{n} w_i \cdot C_i$. Intuitively, the more important a customer is, the sooner Barr wishes to complete the customer's order.

Design an $O(n \log n)$ algorithm to solve this problem to help Barr. You are given a set of n orders with a processing time t_i and a weight w_i for each customer i (assume t_i, w_i are positive integers). You want to decide an ordering of the customer orders so as to minimize the weighted sum of the completion times. Prove correctness of your algorithm (i.e., optimality) and analyze its runtime.

Solution:

An optimal algorithm is to process orders in decreasing order of w_i/t_i. We prove the optimality by an exchange argument.

Consider any other schedule (of customer orders). This schedule must contain a pair of orders i, j such that i comes before j in the alternate solution, and j comes before i in our greedy solution. Further, there must be an adjacent such pair i, j (if not, then no such pair exists, so the alternate solution would agree with our greedy solution). For this pair we have $w_j/t_j \geq w_i/t_i$ by the definition of our greedy solution.

Suppose we swap i, j in the alternate schedule. Let C be the completion time of the job before i (if i is the first, $C = 0$). Note that the completion time of all other jobs remain the same before/after the swap. Before the swap, the contribution of i and j to the weighted sum was $w_i(C+t_i)+w_j(C+t_i+t_j)$, while after the sum it is $w_j(C+t_j)+w_i(C+t_i+t_j)$. The difference between the value after the swap, compared to the value before the swap is $w_it_j - w_jt_i$. Since $w_j/t_j \geq w_i/t_i$, this difference is non-negative, and the swap does not increase the weighted sum (of the alternate solution).

Starting from an arbitrary schedule, we can iteratively perform the swaps until no more such pair of orders remain – then the weighted sum would agree with that of the greedy solution; note that the number of swaps is bounded above by $O(n^2)$ as there are $\binom{n}{2}$ pairs of orders. This shows that the greedy solution is optimal because its weighted sum is no greater than any other schedule.

Note: You should argue why greedy solution is optimal. It is wrong to conclude optimality of greedy just because swapping any pair of jobs in the greedy solution makes it worse (which simply shows that swapping does not improve greedy solution, not its optimality). As we did above, one should argue that greedy solution is no worse than any other solution.

4 (12 points) Attack on Penguin

Do you know that there is a Bear Island in Antarctica? Barr the Bear has landed on Bear Island and is looking to invade Poe the Penguin's homeland. He has created bases in n islands, including Bear Island (we will label islands from 1 to n, where Bear Island is labeled as 1). Barr has also constructed m bidirectional transit lines between the islands for his covert operations. The line connecting island i and island j takes w_{ij} minutes to traverse and (coincidentally) costs w_{ij} Barr coins per hour to maintain. For each of the transit lines, $w_{ij} > 0$. Let d_i be the shortest time it takes to traverse from Bear Island to island i.

Barr the Bear spent too much money on honey and is running low on coins. He decides that he wants to close down some transit lines to save coins. However, he still wants to ensure that closing down transit lines will not affect the shortest amount of time to travel from Bear Island to each island (that is, even after closing down transit lines, d_i would still be the same).

Given n islands, the list of m transit lines, and their costs, help Barr the Bear find a subset of transit lines such that the cost of maintaining the lines is minimized (the total number of coins to be spent per hour), and yet the time taken to travel from Barr Island to every other island remains the same as before closing down any lines.

Solve the problem in $O(m+n\log n)$ time. Your algorithm should output which transit lines are remaining (or which ones to be removed, equivalently). Prove correctness and analyze runtime. If you are using a graph formulation, remember to explicitly state what is the graph you are using (e.g., what are the vertices and edges).

Solution:

Recall that after we run Dijkstra's algorithm we obtain an SSSP-tree by keeping track of predecessor of each node (π). Clearly any SSSP-tree preserves the shortest-path distance from s to every node, and we wish to find an SSSP-tree with minimum sum of edge weights. Using any predecessor does not guarantee an optimal tree (consider $V = \{s, x, y\}$ with $w(s, x) = 1$, $w(s, y) = 2$, and $w(x, y) = 1$). The runtime is the same as that of Dijkstra's.

We do the following modification to Dijkstra's algorithm: When we relax edge (x, y), if $d[x] + w(x, y) < d[y]$, then we update both $d[y]$ and $\pi(y)$ as before, but if $d[x] + w(x, y) = d[y]$ then we check if $w(x, y) < w(\pi(y), y)$ – if this is the case then we update $\pi(y) = x$. This does not change the runtime of Dijkstra's algorithm and does not change its correctness; the same proof from lecture can be used to prove that $d[v] = d(s, v)$ for all v.

We want to show that an SSSP-tree generated by our modified Dijkstra's algorithm is optimal with respect to the sum of edge weights. Let T be the SSSP-tree rooted at s found by our algorithm (and let π be the predecessor function of T) and let T' be another SSSP-tree rooted at s (and let π' be the predecessor function of T') with $w(T) > w(T')$ (for contradiction). Note that $w(T) = \sum_{v\neq s} w(\pi(v), v)$ and $w(T') = \sum_{v\neq s} w(\pi'(v), v)$. Since $w(T) > w(T')$ there exists at least one $v \neq s$ such that $w(\pi(v), v) > w(\pi'(v), v)$. This is a contradiction because our algorithm should choose, among all possible predecessors, the one with minimum edge weight (note that, since T and T' are both valid SSSP-trees by assumption, the s-v path in

T (through $\pi(v)$) and the s-v path in T' (through $\pi'(v)$) must be of the same distance, and our algorithm, in case of such ties, must update the predecessor to the one with the minimum edge weight). Therefore no such v exists, and we conclude that $w(T) \leq w(T')$ for any SSSP-tree rooted at s. This proves optimality of the SSSP-tree found by our algorithm.

5 (0 points) Routing Packets of Fish

Note: This problem will not be graded (please do not submit as we will not read), but it is given as an exercise problem (since we will not cover Network Flow before this homework assignment is released).

Poe and Barr collected so many fish that they have more than they can eat! Therefore, they've decided to ship their fish to other penguin and bear communities across the poles. The n communities form a connected graph connected by a series of m ice chutes. Fish packets can travel across these ice chutes very quickly, but the shipping is limited by the chute's capacity. If communities i and j are connected by a chute, then we can pass $c(i, j)$ fish at once. We say that given a path of chutes, the capacity of the path is the minimum capacity over all the chutes along the path. Poe and Barr want to find paths through the network of chutes that maximize the capacity of fish they can ship.

(a) (0 points) Suppose Poe and Barr set up their shipping depot at community k. Give an efficient algorithm to find for every community c, maximum capacity of any path from k to c. Prove your algorithm's correctness and runtime.

Solution: We want to find a path from k to all other communities m that maximizes the minimum capacity along the path.

$$\max_{p} \min_{(u,v)\in p} c(u, v)$$

We can achieve this by computing running Prim's algorithm from k with the negation of c-values. This will compute a maximum spanning tree. This will take $O(m + n \log n)$ time.

To see correctness, suppose Prim's discovers a $k - m$ path p for some m, but there is a $k - m$ path p' with a larger capacity. Then, this means there is an edge $e \in p$ with $c(e) < c(e')$ for all $e' \in p'$. But discovering p before discovering p' contradicts the exploration rule of Prim's, where we explore by minimum $-c(e)$. Thus, a max spanning tree rooted at k will give the single-source maximum bandwidth paths.

(b) (0 points) Now suppose Poe and Barr want to facilitate trade amongst all of the polar communities. Give an $O(n^2)$ time algorithm to find for every pair of communities, maximum capacity of any path between them. You may assume that chutes are bidirectional with the same capacity in each direction. Prove your algorithm's correctness and runtime.

Solution: Again, we can compute a max spanning tree of the network. Then, for each node $u \in V$, we compute the max capacity by finding the capacity along the max spanning tree to every other $v \in V$ (using BFS or DFS). This will take $O(m + n \log n + n^2) = O(n^2)$ time to compute.

Again, suppose the max spanning tree T contains a $u - v$ path p, but there is another $u - v$ path p' with larger capacity. That is, there is some edge $e \in p$ with $c(e) < c(e')$ for all $e' \in p'$. But consider deleting e from T, and adding some edge $(u', v') \in p'$ to reconnect u to v. This is always possible, because we know that in a spanning tree, every $u', v' \in V$ are connected, but there are no cycles, so there must be some edge along p' that we can use to replace (u, v). But this increases the weight of the max spanning tree – a contradiction. Thus, every $u - v$ path must have maximum capacity.

Bonus (0 points): Suppose Poe is at community p and Barr is at community b and they only care about shipping fish between one another. Give an $O(m)$ time algorithm for finding the path of maximum capacity between p and b.

Solution:

A hint to this problem was given in lecture.